IPST-CNAM
Intranet et Designs patterns
NSY 102
Mercredi 18 Septembre 2013

Durée : **2h 30**
Enseignant : LAFORGUE Jacques

2ème Session NSY 102

1ère PARTIE – SANS DOCUMENT (Durée 1h 15)

1. QCM (35 points)

Mode d'emploi :

Ce sujet est un QCM dont les questions sont de 3 natures :

- **les questions à 2 propositions**: dans ce cas une seule des 2 propositions est bonne.
  - +1 pour la réponse bonne
  - –1 pour la réponse fausse
- **les questions à 3 propositions** dont 1 seule proposition est bonne
  - + 1 pour la réponse bonne
  - –½ pour chaque réponse fausse
- **les questions à 3 propositions** dont 1 seule proposition est fausse
  - + ½ pour chaque réponse bonne
  - –1 pour la réponse fausse

Il s'agit de faire une croix dans les cases de droite en face des propositions.

On peut remarquer que cocher toutes les propositions d'une question revient à ne rien cocher du tout (égal à 0).

Si vous devez raturer une croix, faites-le correctement afin qu'il n'y ait aucune ambiguïté.

N'oubliez pas d'inscrire en en-tête du QCM, votre nom et prénom.

Vous avez droit à **4 points** négatifs sans pénalité.

NOM:	PRENOM:

Un Middleware est :		**1**
1	une couche logicielle, déployé sur le serveur et le client, qui permet de faciliter la programmation de la communication distante entre le serveur et le client	
2	dans une architecture client-serveur, une couche logicielle, utilisée par le client et le serveur pour enregistrer les objets dans une base de données	
3	dans une architecture répartie, le composant logiciel qui permet de répartir la charge sur l'ensemble des composants répartis sur le réseau	

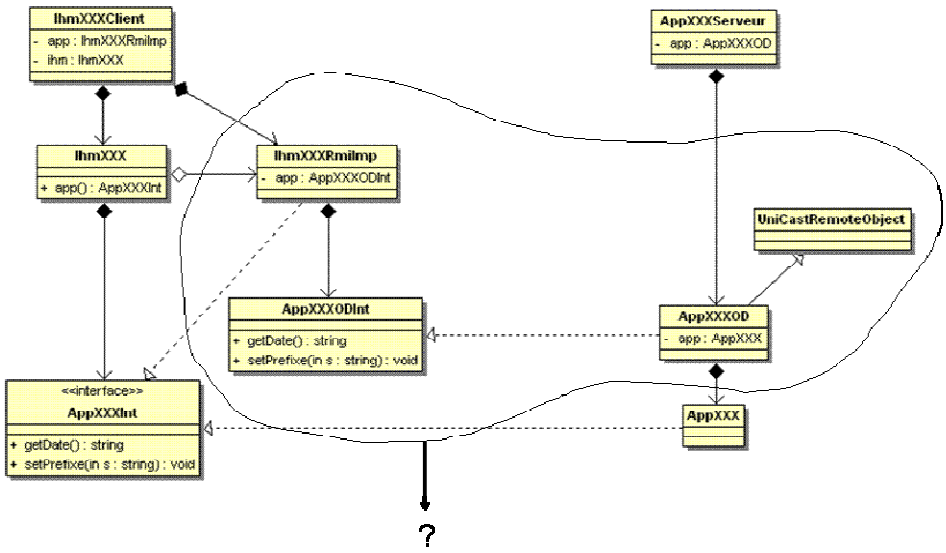
Pour échanger des données entre un serveur et un client, il est indispensable que le serveur et le client soient développés dans le même langage de programmation sinon les données écrites (codage) par l'un ne pourront pas être lues (décodage) par l'autre.		**2**
1	OUI	
2	NON	

L'IDL (Interface Definition Language) est un langage informatique utilisé par les ORB pour faciliter la programmation des interfaces entre les différents composants du réseau qui sont écrits dans des langages de programmation différents.		**3**
1	OUI	
2	NON	

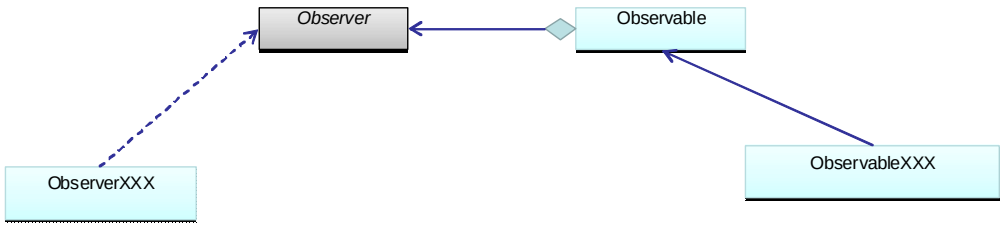
un ORB (Object Request broker) est composé de, au moins : - un annuaire pour enregistrer les objets distribués, - un compilateur idl pour la génération des amorces er des squelettes - une API de classes prédéfinis pour programmer son application distribuée		**4**
1	OUI	
2	NON	

On appelle un objet "distribué" tout objet qui est sérialisé et qui circule sur le réseau		**5**
1	OUI	
2	NON	

En RMI de Java, la classe d'appartenance d'un objet distribué, hérite de RemoteObject et implémente l'interface Remote		**6**
1	OUI	
2	NON	

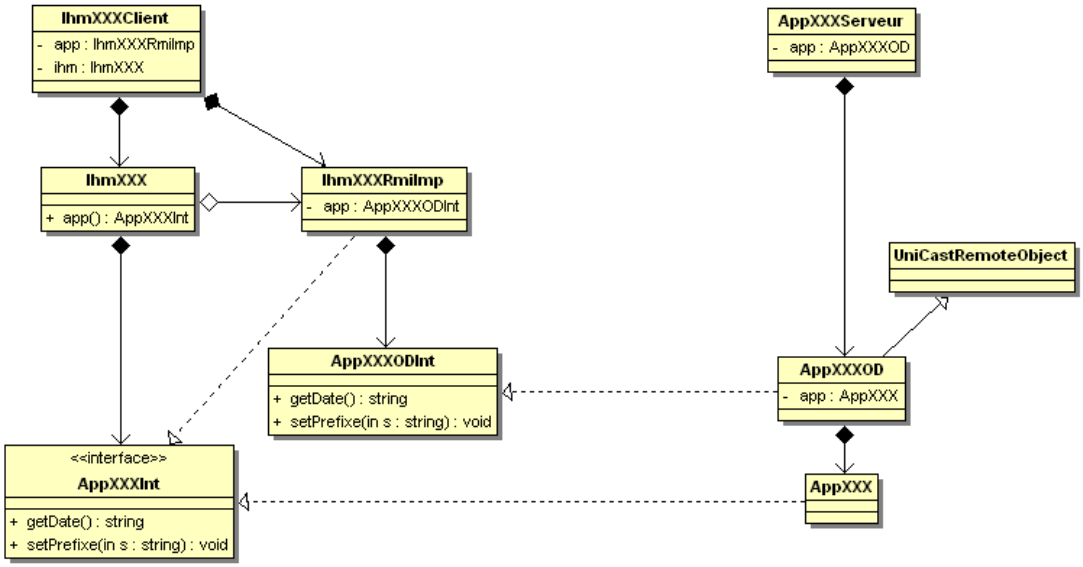
Ceci est le schéma d'architecture de l'atelier 16 (exemple 04) en RMI qui représente une application composé d'un client IHM et de son serveur.  La partie entourée constitue un pont RMI de communication entre l'IHM et l'applicatif		**7**
1	OUI	
2	NON	

Dans une application distribuée, un **factory** est un objet distribué, enregistré dans un annuaire, et dont le rôle est de créer, à la demande d'un client distant, des objets distribués de même nature (appelé "produit"). Le client peut éventuellement récupérer un "stub" (amorce) lui permettant, par la suite, d'utiliser directement t à distance ce produit.		**8**
1	OUI	
2	NON	

Soit le Design Pattern Observateur suivant :  La classe ObserverXXX implémente la méthode update de l'interface Observer qui est appelée par Observable		**9**
1	OUI	
2	NON	

Un client d'un objet distribué distant (OD1) n'a pas besoin de connaitre la localisation physique de OD1 pour l'utiliser		**10**
1	OUI	
2	NON	

Soit la représentation de modèle de classes en UML de l'atelier 16 en RMI vu en cours.		**11**



Le rôle de IhmXXXRmiImp est de :

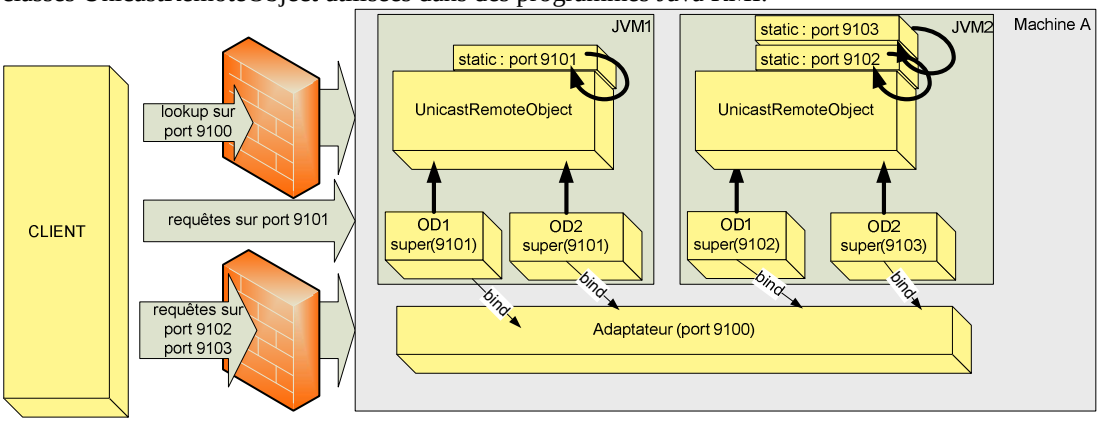
1	implémenter les méthodes distantes décrite par l'interface RMI AppXXODInt	
2	implémenter les méthodes de l'interface AppXXXInt afin de les implémenter sous la forme d'appel de méthodes distantes implémentées par AppXXXOD	

Soit 2 clients (A et B) qui appellent la méthode distante m1 de l'objet distribué OD1.		**12**
1	A et B peuvent appeler en même temps la méthode m1 de OD1	
2	A et B ne peuvent pas appeler en même temps la méthode m1 de OD1 et il faut donc que la méthode m1 soit "synchronized".	

En Java, la méthode **public static Remote toStub(Remote obj)** permet d'enregistrer l'objet distribué 'obj' dans l'annuaire RMI		**13**
1	OUI	
2	NON	

En RMI de Java, les paramètres des méthodes distantes peuvent être :		**14**
1	des éléments de type primitif (int, char, long, byte, ...)	
2	des objets dont la classe d'appartenance est une classe qui implémente l'interface Serializable	
3	des objets de n'importe quelle classe	

Soit le schéma suivant qui représente un fonctionnement possible de plusieurs serveurs de socket des classes UnicastRemoteObject utilisées dans des programmes Java RMI.		**15**



1	Dans la JVM1, on peut créer un nouvel OD qui s'exécute sur le port 9104	
2	On peut créer une nouvelle JVM3 dans laquelle, on crée un nouvel OD qui s'exécute sur le port 9100	
3	Dans la JVM2, on peut créée un nouvel objet distribué RMI sur le port 9102	

En RMI, l'appel d'une méthode distante, entre un client et un objet distribué RMI se fait de la manière suivante : 1/ le client récupère l'amorce (ou stub) de l'objet distribué 2/ le client utilise les méthodes de l'amorce		**16**
1	OUI	
2	NON	

Lesquelles de ces descriptions suivantes sont des modèles de communication asynchrones :		**17**
1	un serveur pousse ses évènements dans une file d'attente par client connecté (intermédiaire), et les clients tirent ses évènements à leur rythme	
2	un serveur pousse son évènement dans un proxy de consommateur, et à son tour, le proxy de consommateur pousse l'évènement au consommateur	
3	un serveur appelle la méthode distante d'un client afin de lui pousser l'évènement. La méthode distante n'est pas void.	

En RMI, chaque objet distribué doit s'exécuter sur un port (TCP/IP) différent afin de ne pas mélanger les canaux de communication entre tous les clients et chacun des objets distribués du serveur.		**18**
1	OUI	
2	NON	

Les "patrons de conception" ou Design Patterns" sont des modèles standard et réutilisables de conception de la solution à la problématique de la réalisation d'une partie d'un logiciel informatique.		**19**
1	OUI	
2	NON	

Dans la conception UML d'une application informatique, le "Canevas" est un Design Pattern particulier qui répond à la problématique de création d'objets graphiques génériques qui doivent dessiner dans un dessin (ou canvas)		**20**
1	OUI	
2	NON	

Le DP Singleton permet de rendre transparent pour le programmeur la création unique d'un objet		**21**
1	OUI	
2	NON	

Soit le code suivant d'implémentation d'un singleton : **22**

```
public class SingletonXXX
{
    static private SingletonXXX sg = new SingletonXXX ();

    private SingletonXXX () { }

    static public SingletonXXX getSingletonXXX()
    {
        return sg;
    }
}
```

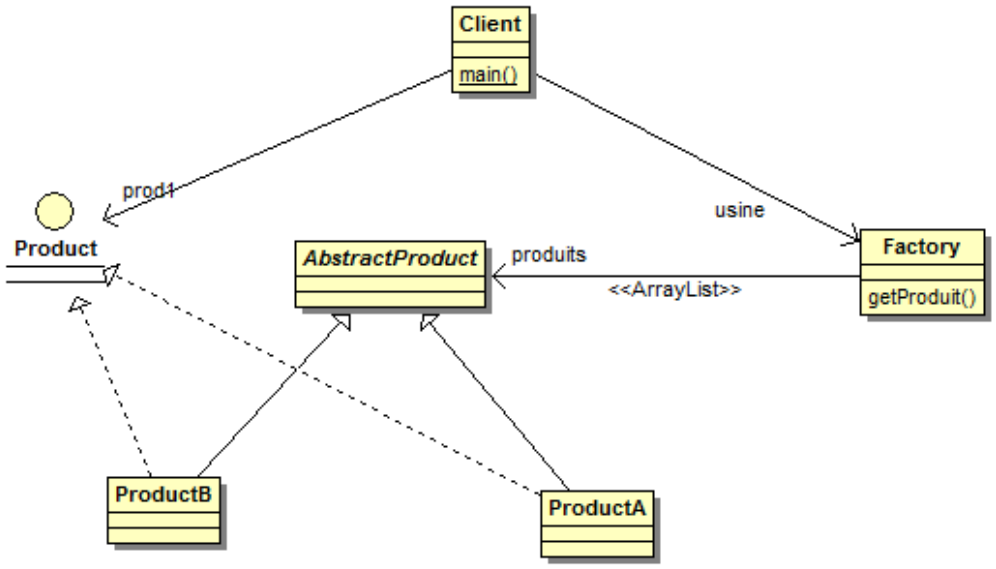
Ce code est correct.

1	OUI
2	NON

Le DP Factory a pour fonction de créer des objets uniques répartis sur le réseau **23**

1	OUI
2	NON

24



Ce DP est celui du Factory.
Le rôle de la classe AbstractProduct est de :

1	faire le new de création des objets fabriqués
2	mettre en commun les attributs de tous les types de produits réels fabriqués par l'usine
3	rendre abstrait l'utilisation des produits par le Client

Dans un DP Factory, le Client doit connaître les différents types de produits créés par le factory **25**

1	OUI
2	NON

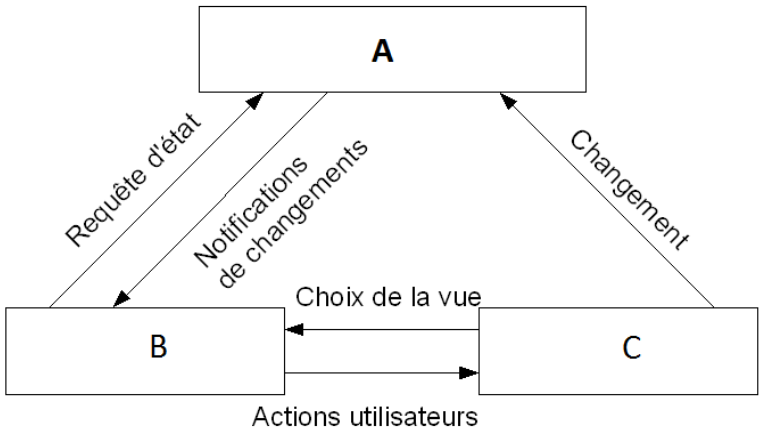
Le DP Builder est un DP utilisé dans celui de Singleton afin de construire l'instance unique par assemblage d'autres singletons **26**

1	OUI
2	NON

Le rôle général d'un Adapter est de s'adapter à des interfaces et des classes qui utilisent ces interfaces, sans les faire évoluer (car cela pourrait être pénalisant ou tout simplement parce que l'on n'a pas le code source).		**27**
1	OUI	
2	NON	

L'adaptateur et l'adapté implémentent la même interface		**28**
1	OUI	
2	NON	

29



Ce schéma représente le DP Modèle-Vue-Controleur.
Les lettres A, B et C sont définies ainsi :

1	A = Modèle; B=Vues ; C= Controleur	
2	A=Controleur; B=Vues; C=Modèle	

Un Proxy est un DP dans lequel deux classes A et B héritent d'une même classe C		**30**
1	OUI	
2	NON	

Le DP Iterator permet de parcourir des collections d'objet quelques soient leurs implémentations.		**31**
1	OUI	
2	NON	

Le DP Observateur est implémenté en Java via la classe Observable et l'interface Observer.		**32**
1	Le rôle de la classe Observable est de notifier un autre objet (appelé observateur) de certains évènements	
2	Le rôle de la classe Observable est de contenir la liste de tous les observateurs	
3	Le rôle de la classe Observable est de consommer les évènements notifiés par un un autre objet qui implémente l'interface Observer	

Laquelle des descriptions suivantes est un principe de communication synchrone :		**33**
1	le producteur dépose à son rythme ses évènements dans une file. Le ou les consommateurs peuvent alors récupérer ces évènements	
2	le producteur pousse ("push") chaque évènement vers chacun des consommateurs via une méthode distante qui retourne un état de consommation	

Un proxy est une classe se substituant à une autre classe. Par convention et simplicité, le proxy implémente la même interface que la classe à laquelle il se substitue.		**34**
1	OUI	
2	NON	

Le DynamicProxy est un DP qui permet d'exécuter dynamiquement une classe Java (Le ClassLoader de Java est implémenté suivant ce DP)		**35**
1	OUI	
2	NON	

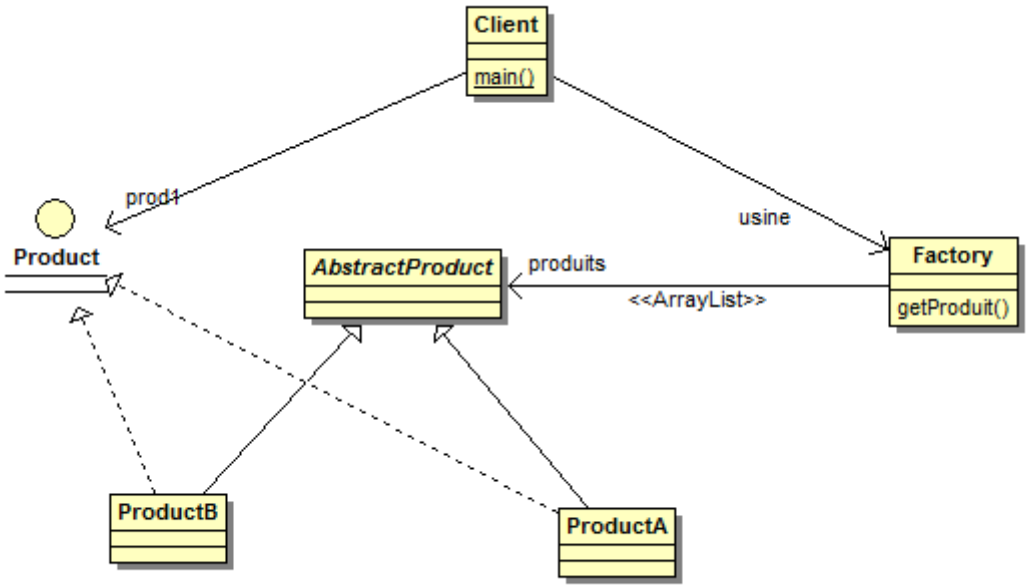
Tourner la page

2. Questions libres (15 points)

Chaque question est notée sur 7,5 points.
*Vous répondez à ces questions sur une **copie vierge double** en mettant bien le numéro de la question, sans oublier votre nom et prénom.*
Vous mettez le QCM dans la copie vierge double.

QUESTION NUMERO 1

Soit le schéma suivant qui représente le design Pattern du Factory :



Commentez le schéma ci-dessus (Définition, principes, rôles de chaque composant, ..).

Dans un exemple concret (donner un vrai nom pour A et B, et pour Factory), écrivez en Java un exemple de code de Client qui crée le factory et l'utilise.

QUESTION NUMERO 2

Nous avons sur le réseau deux objets distribués RMI (Java), appelés Entreprise et Client, qui doivent s'échanger des factures et des bons de commandes. La communication de ces informations est asynchrone.

Expliquez votre choix d'architecture pour faire la communication des informations.
Décrivez sous la forme UML votre choix de conception.

(Fin de la 1ère partie sans document)

Si vous rendez la copie de cette 1ère partie, vous pourrez commencer la 2ème partie.

2ème PARTIE – AVEC DOCUMENT (Durée 1h15)

3. PROBLEME (50 points)

Nous envisageons de réaliser un système d'information Intranet qui permet de jouer au Poker en ligne.

Un joueur est un objet unique d'un programme JAVA appelé Joueur. Ce programme Java prend en entrée le nom du joueur.
Chaque joueur exécute ce programme qui se connecte à une table de poker qui est un objet distribué unique sur le réseau. Le nom de la machine sur laquelle s'exécute la table de Poker s'appelle "POKER001" et le nom de l'OD est "TABLE_POKER".

Le programme Joueur crée une IHM appelée TablePokerIhm qui permet à chaque joueur de visualiser la table de poker et permet de passer des commandes opérateurs vers la table de Poker

---

1/ Faite le schéma d'architecture logiciel (ou schéma UML d'Acteur) de votre solution : réseau, machines, composants (ou acteurs), fonctions (transitions entre les composants), ...
Précisez le rôle de chacun des composants. Justifiez vos choix.

2/ Faire le diagramme de classe UML de chacun des composants.
Mettez en évidence les méthodes et les attributs importants. Précisez en marge des diagrammes le rôle de chacune des méthodes.

3/ Mettez en évidence les Designs Patterns que nous avons vu en cours qui se retrouvent dans vos diagrammes de classe. Commentez.